Information Management in Intelligent Environments

by

Ruben E. Brown

S.B., Massachusetts Institute of Technology (1999)

Submitted to the
Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
February 2001

Copyright 2001 Ruben E. Brown. All rights reserved.

The author hereby grants to M.I.T. permission to reproduce and distribute publicly paper and electronic copies of this thesis and to grant others the right to do so.

Author............................... Department of Electrical Engineering and Computer Science
February 1, 2001

Certified by............................ Howard E. Shrobe
Principle Research Scientist, MIT Artificial Intelligence Laboratory
Thesis Supervisor

Accepted by............................ Arthur C. Smith
Chairman, Department Committee on Graduate Students
Information Management in Intelligent Environments

by

Ruben E. Brown

Submitted to the Department of Electrical Engineering and Computer Science on February 1, 2001, in partial fulfillment of the requirements for the degree of Master of Engineering in Electrical Engineering and Computer Science

Abstract

In this thesis, I designed and implemented a system for handling information in complex computational environments. The system addresses the needs for context dependent information handling and display in an intelligent environment. The system is used as part of the Metaglue development environment to drive the information display in the Intelligent Room.

Thesis Supervisor: Howard E. Shrobe
Title: Principle Research Scientist, MIT Artificial Intelligence Laboratory
Acknowledgments

I would like to thank Howie Shrobe, my advisor, his advise and wisdom has been a constant and unfailing guide. I would also like to thank all the assembled masses of the room hackers who were amused by my antics. In particular, I should thank Stephen Peters for being my occasional sounding board and the only other married man under the age of fifty in the lab. In addition, I would like to extend my gratitude to my friends and family without whose support I could not have pulled this off. Finally, I must thank my wife for her love.
Contents

1 Introduction .............................................. 10
   1.1 Motivation ............................................ 11
   1.2 Background .......................................... 11
      1.2.1 Problems with Corporate Memory ................. 11
      1.2.2 Previous Research ................................. 12
      1.2.3 The formal language of APT ....................... 15
      1.2.4 SAGE ........................................... 16
   1.3 Putting it all together ............................... 18
   1.4 Goals ................................................ 18
   1.5 Project Description .................................. 19
      1.5.1 Mechanisms for Structuring data ................ 20
      1.5.2 An Adaptive Transform Engine .................... 20
      1.5.3 Interface for User Interaction and Control .... 21
   1.6 An Example Interaction .............................. 21
      1.6.1 Non-integrated Response ......................... 22
      1.6.2 Integrated Information System Response ....... 22

2 Design .................................................. 24
   2.1 Information Representation, Information Containers .... 24
   2.2 Translation Infrastructure, Information Functions .... 25
      2.2.1 Complex Context based Translation ............... 25
   2.3 Communication Infrastructure ........................ 27
A.8  InfoProducer ........................................... 58
A.9  InfoProducerAgent ........................................ 59
A.10 InfoManager .............................................. 63
A.11 InfoManagerAgent ......................................... 64
A.12 MetaManager ............................................. 68
A.13 MetaManagerAgent ....................................... 69
List of Figures

2-1 Information Function behavior: Note that the infofunction only changes the type of the container, but leaves the contained information the same.

2-2 Information Flow: In the simplest possible communication, there exists only an information producer and an information consumer. The producer acts as a source of information containers and the consumer as a sink.

2-3 Information Manager behavior: Here a manager is carrying out the role of a translation device, converting one type of information container to another type in order to meet a set of display constraints.

3-1 Infosystem Diagram: The implementation structure of the infosystem, showing the Java class interfaces and their implementing classes.

3-2 The InfoContainer: The InfoContainer implements the possesable interface and provides a series of methods for comparison of Information types. The view method provides a mechanism for directly applying InfoFunctions to the InfoContainer.

3-3 The InfoFunction: The InfoFunction provides the view method, taking an InfoContainer of some type and producing one of another.

3-4 Summary of InfoHandling Interfaces: This figure presents the publicly accessible methods of the InfoHandling Agents.
3-5 The InfoProducer: the InfoProducer interface is implemented by the InfoProducerAgent. The InfoProducerAgent a set of useful protected methods to all subclasses. ................................. 37

3-6 The InfoConsumer: InfoConsumerAgent implements the InfoConsumer interface, providing protected methods for maintenance of the consumes methods. ................................. 38

3-7 The InfoManager: InfoManagerAgent implements the InfoManager interface, in order to maintain correctness of the broadcast receipt it extends InfoConsumerAgent. However, it implements the protected methods from InfoProducer for ease of extension. ................. 39

3-8 MetaManager GUI: This gui provides direct interaction to the infosystem. The InfoContainer and active InfoHandlers are all listed. In addition, the Feed button allows manual user direction of any info container to any consumer. The link button allows the user to connect Infoconsumers to InfoProducer, automatically registering the consumer for all the appropriate containers the producer produces. ............. 41
List of Tables

1.1 Cleveland and McGill’s ranking of the accuracy in the perception of quantitative information through different graphical intermediaries. Reproduced from Jock Mackinlay’s thesis. [Mackinlay, 1986] .......................... 16

1.2 Mackinlay’s extension of Cleveland and McGill’s ranking of perception tasks: Mackinlay’s separation of information into Quantitative, Ordinal, and Nominal sets provides the basis for evaluating the effectiveness of any given graphical language. Reproduced from Jock Mackinlay’s thesis. [Mackinlay, 1986] .................................................. 17
Chapter 1

Introduction

In a world where turn over in the work force is increasing and the average amount of time a worker stays at a given corporation is decreasing, corporations face a growing problem preserving their corporate memory. This largely oral tradition preserves design rationale over time allowing workers access to the knowledge of their predecessors. Software systems are trying to provide a mechanism through which design rationale can be preserved over time. However, often the labor cost of using these systems far outweighs their benefits. This paper proposes a system for making these automated information gathering systems more useful by reducing the labor cost of extracting information from the system. In order to allow the user of a computerized information service to most efficiently gather information, the computer needs to be able to take advantage of this power of different representation systems to provide emphasis to and speed absorption of the desired information. This thesis project produced a tool that will make this more possible.

This thesis project gives the intelligent room the ability to deal with abstract information for the purpose of display. This was accomplished by designing and implementing an Application Programming Interface (API) for dealing with information handling within metaglue, a software agent environment.

This package then is an extension of the metaglue programing environment allowing sophisticated handing of information display in such intelligent environments as
1.1 Motivation

Corporate memory is a phenomenon where information about projects and designs is preserved even after the personnel working on the project have changed entirely. The main avenues through which corporate memory is transmitted are the oral tradition and documentation. To facilitate the transmission of this corporate memory, automated information gathering systems have been developed. The primary problem with these automated information gathering systems is that the amount of effort necessary to input data often outweighs the benefits of the information system.

Previous research has concerned itself primarily with reducing the amount of effort necessary to input useful information. Although research continues in this direction, additional research is being conducted to improve the benefits provided by the system by creating synthetic information from sparse input and contextual queues. Future research is needed to investigate methods for extracting information from the system, and assisting the user in gathering, understanding and presenting this information. Such tools will dramatically improve the benefits of automated information systems.

1.2 Background

1.2.1 Problems with Corporate Memory

Corporate memory is a phenomenon where information about projects is preserved even after the personnel working on the project have changed entirely. The main avenues through which corporate memory is transmitted are documentation and oral tradition. Documentation is a very good tool for preserving information about specifications of a final product; however, documentation is not good at preserving design rationale. Supplemental information about designs is often only preserved through oral tradition.
At current, few tools exist for capturing the knowledge preserved in the oral tradition. In the past, capture of design rationale was done through a system of formal design reviews. However, with the mass proliferation of computers in the technical workplace, e-mail status reports and quick turn around correspondence are replacing these formal reviews.

E-mail is only a limited tool for capturing design rationale. Unlike a formal design review, e-mail serves only as an extension of the oral tradition and does not lend itself to structured presentation of design rationale. Instead, e-mail is an informal medium that often predicates previous involvement in the dialogue for understanding information regarding design rationale.

Understanding this design rationale is necessary to the maintenance and continuing development of products. The tools that exist in industry to extract design rationale from a loosely structured dialogue are vastly inadequate. The formal design review is becoming a thing of the past; the informal discourse that has replaced it is not intended for an audience outside the technical sphere and it never captures the majority of the design rationale of a project. A tool for enhancing these informal reports with a better mechanism for capturing, organizing and representing design rationale will prove invaluable.

1.2.2 Previous Research

Work continues at MIT to find a solution to the problems of using automated tools to preserve corporate memory. Human Computer Interaction: Intelligent Environments initiative at the AI lab has spawned a number of research projects on this topic. Current research goes on to reduce the cost of entering data and to expand the amount of information included in computerized histories. However, information capture is only one side of the coin. The flip side is information representation and display. Together these two elements build a complete information system.
Evaluating Information Systems

In order to understand the importance of adaptive information systems it is first necessary to develop a theory for evaluating the total worth of a piece of information. Let us define the effort necessary to understand a given piece of information as a quantity of man-hours as $\epsilon_u$, the effort necessary to input that information into the system as $\epsilon_i$, and the value of that piece of data in man-hours of work saved by not having to replicate previous work as $V_i$. From these three quantities we can define a simple heuristic that determines the total value in man-hours, $V_t$, of a given piece of information. This yields the equation included below:

$$V_t = V_i - \epsilon_i - \epsilon_u$$

(1.1)

In many automated information gathering systems the amount of labor saved by a piece of information is far less than the time lost in capturing and inputting. However, even in a system where $\epsilon_i$ is eliminated by an ideal input mechanism, the effort necessary to understand and utilize the information can still reduce the total value significantly.

In order to maximize the $V_t$, we must be able to reduce the effort necessary to understand, assimilate and utilize information. To do so we should look at the way in which learning takes place. Simply put, everyone perceives the world a little differently than the next person. For example, some people find images easier to assimilate than text while others find that textual resources are preferable.

Current information systems make the assumption that a single representational system can satisfy the learning needs of all of its users. Producing a system for different representations that can adapt to an individual user’s needs will greatly reduce $\epsilon_u$ and allow the user to get the maximum value out of the information system.
Information Input

Reducing the cost inputing information is the subject of a number of research projects. At the AI lab, Luke Weisman’s research is looking into making capturing the oral tradition easier. Under the tutelage of Professor Randy Davis, he is developed an interactive sketch pad for design of mechanical apparatus. This pad interacts with a designer as he constructs his device, asking him for information about his design choices and providing help clarifying and preserving his diagrams. [Weisman, 1999]

Wiesman’s project produces an unstructured history including both design and an associated rationale. This history is being examined by Christine Alcarado with an eye to providing more than just enhanced diagrams. As the focus of her master’s thesis [Alvarado, 2000] she developed a natural sketching mechanism that attempts to understand this unstructured history and from it create demonstrations of the mechanical design in use. Further research in this area is the subject of another master’s thesis, that of Michael Oltmans, which sought to extend the scope of the information capture by includeing gesture and verbal explanation as part of the design history. [Oltmans, 2000] [Oltmans, 2001]

All three of these systems use user feedback reduce the amount of effort the designer needs to input a large body of information into an information system. By making assumptions about what the user means, and then verifying them, these systems can synthesize information that the user would otherwise have to enter into the system. This reduces greatly the effort of capturing important design information from large scale data entry to the easier task of vetoing incorrect assumptions by the information system.

Information Representation & Display

Previous research has built up a body of work concerning the representation and display of information. The field of automated presentation tools is not new. In 1986 Jock Mackinlay completed a doctoral thesis at Stanford in which he developed
a formal basis for computer generation of information presentations. He produced a
prototype system, APT (A Presentation Tool) which while limited served as a proof
of concept. Since his thesis [Mackinlay, 1986], a family of systems built his formal
language have appeared.

Steven Roth has built an initiative centered at Carnegie Mellon University to
explore the expansion of Mankinlay’s language. The System for Automated Graph-
ics Explaination (SAGE), in particular, is a sophisticated realization of Mackinlay’s
thesis. [Joe Mattis and Steven F. Roth, 1991] [Mei C. Chuah et al., 1995] Since the
creation of SAGE Roth has been involved in the creation of user interface tools such as
Visage which utilize SAGE and modern user interface pradymes to create complex in-
formation centric dynamic user interfaces for information. [Steven F. Roth et al., 1996]
Current research at CMU is focusing on the creation of a system analogous to SAGE
which would to handle time varying multi-media data.

A large body of pre-existing research into cognative preception of graphics and
semiology supports APT. In particular Bertin’s seperation of graphical objects and
relationships provides a basis for the formalisms of Mackinlay’s descriptive language.

1.2.3 The formal language of APT

The formal language of APT is based in the metaphor of the graphic sentence. Each
graphic sentence encapsulates the set of relations that define the a single graphic
expression. All the well-formed sets of graphic sentences define a graphics language
for a given domain of information. A given set of facts is consided to be expressible
in a given language; if the language contains a sentence that encodes all the facts in
the set, it encodes only the facts in the set. [Mackinlay, 1986]

Having exstablished a basis for evaluating the expressiblity of a given set of facts in
a given language, Mackinlay defines a set of criterion for analysising the effectiveness
of a given language at expressing a given set of facts.

Cleveland and McGill’s 1984 paper provides empircal evidence for the fixed or-
<table>
<thead>
<tr>
<th>Most Accurate</th>
<th>Relative Position</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Length</td>
</tr>
<tr>
<td></td>
<td>Angle or Slope</td>
</tr>
<tr>
<td></td>
<td>Area</td>
</tr>
<tr>
<td></td>
<td>Volume</td>
</tr>
<tr>
<td>Least Accurate</td>
<td>Color or density</td>
</tr>
</tbody>
</table>

Table 1.1: Cleveland and McGill’s ranking of the accuracy in the perception of quantitative information through different graphical intermediaries. Reproduced from Jock Mackinlay’s thesis. [Mackinlay, 1986]

Ordering of different graphic representations, seen in Figure 1.2.3, based on the accuracy with which qualitative information is perceived using that representation. [William S. Cleveland and Robert McGill, 1984] From this Mackinlay conjectures that in the domain of 2-D graphics there exists a fixed ranking of the effectiveness of rendering techniques for all types of information. In order to extend this ranking of graphical intermediaries, Mackinlay separated all information sets into either qualitative, ordinal, or nominal and postulated a fixed ranking for each sub-domain of information see figure 1.2.3. He adds further specificity to the system by postulating the “Principle of Importance Ordering” that states that more important information is always encoded using a more “effective” method than less important information.

Having defined a set of tools for evaluating a set representation, Mackinlay defines a tool for generating multiple designs. To do this he defines a set of composition operations. These are built from a set of primitive graphic languages. His rankings continue to be useful tools in building other automated representation and display tools.

1.2.4 SAGE

Sage, developed by Steven Roth at CMU, provides an extended set of generalizations built on the basic framework of APT’s formal language. Sage adds finer grain distinctions than the tripartite sub-division of information utilized in APT. SAGE makes
Table 1.2: Mackinlay’s extension of Cleveland and McGill’s ranking of perception tasks: Mackinlay’s separation of information into Quantitative, Ordinal, and Nominal sets provides the basis for evaluating the effectiveness of any given graphical language. Reproduced from Jock Mackinlay’s thesis. [Mackinlay, 1986]

A distinction between data that represents amounts and that which represents coordinates. In addition, SAGE defines domain membership for information, creating the domain specific conventions for time, space, temperature, and mass. This allows SAGE to define graphical conventions for given sets of information, such as displaying time along the horizontal axis. [Mark Derthick and Steven F. Roth, 2000]

Having added these additional criterion to effectiveness evaluation, SAGE expands the evaluation of Relational-Structure. Mackinlay’s APT had a simple mechanism for distinguishing sets with functional dependency from those without it. SAGE employees analysis to determine a more sophisticated characterization of the data.

Beyond using the above characterizations of information to determine acceptable representational languages for use with a given data set, SAGE implements a set of tools for better representing complex data types. APT decomposes compound informational relationships into simpler relations which it then represents through the simple relational evaluation mechanism. SAGE is able to treat these compound sets
as single complex data set. Continuing research by Roth has led to the development of VISAGE, with provides additional power with information centric user interface design tools. [Mei C. Chuah et al., 1995] [Steven F. Roth et al., 1996]

1.3 Putting it all together

The current research into information capture, representation and display is promising. However, it lacks one key element, a context. Both capture tools and display tools operate in a vacuum. The information generated concerning the context of the interaction with an information display engine is lost, even though it is captured by the existing input tools. Capture and representation are both functions of an intelligent environment. The Intelligent Room provides the prefect context to attempt the integration of these to distinct types of system.

1.4 Goals

The purpose of this thesis is to investigate mechanisms to assist the user in assimilating and understanding large amounts of loosely structured information, such as the history of a design process. Such mechanisms for organizing information would prove an invaluable aid for making corporate memory less of an abstract concept and more of a concrete resource for continuing design efforts.

The goal of this project is to produce a tool for structuring and presentating information collected in an intelligent environment. The user’s requirements constrain the design of this tool; it should be dynamic, adaptive and sophisticated. In order to be dynamic it must support changes in representation of data once the information has initially been processed. To be adaptive, the tool must be able to change representational system to better serve a user’s needs, allowing him or her to absorb information through his or her preferred medium and information structures. Furthermore, this adaptation mechanism must be sophisticated enough to recognize and separate the
multi-faceted interests of a single user.

Within these constraints, the body of work proposed for this thesis project should result in a tool for constructing abstract and rational organizations of information from an unstructured or loosely history of design decisions and rationale. The design of this tool can be broken down into three subsections:

1. The design of a mechanism for creating structure from unstructured or loosely structured information with contextual queues

2. The design of an engine for transforming information from one structured representational system to another

3. The design of an interface for user control of the adaptive the transformation engine

To limit the scope of this research project, this thesis project focused on the creation of a mechanism for information transform. Mechanisms for structuring data and adapting to user preferences will be addressed only to the extent that they are necessary to validate the usefulness of the transform mechanism.

1.5 Project Description

This thesis project concerned itself with the creation of a tool that allows information to be viewed through multiple representations. In order to realize such a tool in software three distinct software components are necessary. First, the system must have a tool for receiving information and converting information in to an abstract data structure. Second, the system must possess a tool to convert this abstract data structure into a human compatible representation. Third, there must be a user interface with at least a rudimentary feedback mechanism to allow the user to specify the representation they would find most useful at any given time.
1.5.1 Mechanisms for Structuring data

Over the course of time the tools this project provides will allow creation of mechanisms to assimilate more complex and less structured information. However, at the inception of the project only simple information input are needed. As the projects API comes into heavier use these mechanisms will increase in complexity.

1.5.2 An Adaptive Transform Engine

The core of this software system is an engine that lets a computer manipulate human data structures. Computers are already able to manipulate the highly structured abstract data structures which we create, such as tree-structures or lists. However, these structures do not map directly to human data structures such as essays or oral presentations.

In addition, computers can only use data structures to manipulate information, they are unable to convey meaning to humans through their choice of structure. Humans are able to use different forms of presentation to attach meaning to information. Different forms of representation of the same set of information can create different emphasizes and provoke the audience to absorb different subsets of the total body of information presented.

That information systems cannot yet do this forces the effort to understand information to be higher than it need be. Equation 1.2.2 explains how increased effort to understand information in the system results in a decrease in the total usefulness of the information stored. To increase the value of information stored in the system you must be able to decrease the effort necessary to extract and understand the information. If the information service could adapt to present information to the user in the form her or she could best understand it, it would greatly reduce the time necessary to extract useful information.

The answer to this dilemma is to devise a system where the representation transformation engine deals only with concrete formulaic abstraction, while the user is
shown information in human compatible data structures. The system manipulates this concrete abstraction to create the transformations from one human representation system to another, thus creating the perception that it is going through the same process as the human mind would if trying to best teach the user the information.

1.5.3 Interface for User Interaction and Control

There are a wide range of possibilities for how to setup a user interface to the engine. In this case we are interested in giving the user control over what information they are accessing and over the form in which this information is presented to them. The simplest of interfaces allows the user to pick a presentation style to go along with each piece of information. In the long term this interface should develop into a complex system of its own. An ideal system would construct an “optimal” representation for data using a history of the user’s preferences concerning similar topics. However, such an adaptive system needs to be able to recognize that a user may want to approach the same data from different perspectives; trying to tie a single representation to a single piece of data can prove too restrictive. Over the course of time, the project API allows for the extension of the user interface so that it can fulfill these more complex functions, beyond the basic display requirement.

1.6 An Example Interaction

This section describes an example the type of interactions resulting from the integration of the context input tools and the intelligent representation mechanism. One of the primary uses of the intelligent room is as conference space. As such meetings occur on a regular schedule in the Intelligent Room, with a list of expected attendees. Here is an illustrative example of how a group meeting might progress.

Jim, a research scientist, is leading a meeting at which he presents the research of a group competing with his own for a grant. On the overhead projector, he is displaying a web-page of one of the opposition’s papers. Upon seeing the web-page,
Jim’s boss immediately stops him and tells him that the grant coordinator needs this information immediately. In the context of this meeting the information is best displayed by a projector pointed at the wall behind Jim. However, Bob, the grant coordinator, is not at this meeting.

1.6.1 Non-integrated Response

In a world without capture of contextual cues, the process for getting the information from the meeting to the grant coordinator is complex. Jim stops the meeting, brings up an e-mail program, copies the URL out of the browser and pastes it into the message, find out who is the grant coordinator for the group and then looks up Bob’s e-mail address and finally sends the e-mail.

1.6.2 Integrated Information System Response

In a world with capture of contextual cues, the process becomes much simpler. Jim is standing next to the wall on which the web-page is displayed, points at it as says “Send this to the grant coordinator,” and the URL is bundled up and sent to Bob.

This simplified interaction is made possible by several layers of intelligence in the environment.

1. Speech to text processing

2. Natural language processing done to attach meaning to the text

3. Determination of which grant coordinator Jim was referring to through contextual cues, such as checking with group is having a meeting, a history of the dialogue, or a request for clarification

4. Determine that the information to be sent is the web page Jim is pointing at, using another set of contextual cues including vision processing that recognizes pointing or a laser pointer dot
5. The processing of the web-page and the sending of an email container a URL to Bob, the group’s grant coordinator

Steps one through four are implemented to differing degrees through the existing agent system. [Michael Coen, 1997] [Michael Coen, 1999] [Michael Coen et al., 2000b] However, until information representation and display are integrated, step five can only be done in an ad hoc manner. The abstract information architecture makes it possible for the Intelligent Room to reason about the information contained in the web page and establish that a web-page can be represented as a URL in an e-mail message.

Furthermore, an information management agent working on behalf of Bob can receive a notification from Jim’s agent that a web-page needs to be displayed to Bob. The choice of sending Bob an e-mail then happens automatically because Bob’s agent knows that Bob is not near a display on which it could display a web-page, and it knows that a web-page can be represented by an e-mail. It then sends e-mail containing the representation of the web-page, in this case a URL.

This architecture allows other representations to happen automatically as well. A good example of this is the display of tabular data as graphs on different display devices. This architecture allows Jim to tell the room “Show me the competitors stock quotes from the last six months.” Jim’s information management agent then generates a chart automatically taking into account the displays available to Jim. This chart is generated from an up to date snapshot of the data. The Infosystem architecture allows a level of automation that removes the risk that information for a presentation will become unusable because of a change in venue and the risk that the information will become dated.
Chapter 2

Design

The core of this API rests on 5 major abstractions: the information container, the information function, the information producer, the information consumer, and the information manager. These abstractions are separated into three categories, Information Containers provide the abstract representation of information within the system. Information Functions provide the basic mechanism for operating on Information Containers. The remaining three are responsible for the flow and communication of information through the system.

2.1 Information Representation, Information Containers

At the root of the design is an abstract representation of Information. This representation is the Information Container. Each information container represents a piece of data of a specific type. Typing the data allows the system to reason about general categories of information rather than specific instances. Furthermore wrapping the instance of a specific piece of data inside an information container allow the system to act on information that is not directly representable in the programming language, such as analogue streamed video like the output of a VCR.
In theory this abstraction makes possible the implementation of just in time information handling, where reasoning and representation choices can be made without knowledge of more than an abstract description of the data. For example, reasoning for representation of an SQL table could occur while the information was still being pulled from the database.

2.2 Translation Infrastructure, Information Functions

Information functions provide a mechanism for providing translation services for information. Each information function is limited to producing a single type of Information Container, but is capable of taking a number of different types of information container as input, see Figure 3.3. This unifies the translation mechanism. By simplifying code maintenance in this way the correctness of a translation mechanism is more likely to endure as development of the system continues. Providing this concrete mechanism for information translation, separate from the messaging system, reduces the dangers of ad hoc re-implementation.

2.2.1 Complex Context based Translation

Within this simple structure of information function it is possible to implement complex representation mechanisms. For example, a given set of pre-conditions can be feed to an infomation function at its construction. These pre-conditions provide a context for the function’s translation mechanism. One such use of context based translation control is the creation of graphical representation of structured data.

Using an the Java Expert System Shell, an information function for translating tabular data into charts and graphs was designed. This expert system extends the functionallity of systems like SAGE and APT by using their formal tools to evaluate the worth of a given choice of representation while at the same time encorperating
Figure 2-1: Information Function behavior: Note that the infofunction only changes the type of the container, but leaves the contained information the same.

Domain knowledge of the display environment. The Intelligent Rooms resource managing agent is able to provide the pre-existing context for any given display, including information about hardware specifications such as pixel depth and resolution as well as meta-data about the devices such as physical location. [Gajos, 2000]

An expert system can use this context to better develop a representation for any given set of data. This representation engine creates charts from tabular data using a series of rules. The first set of rules determines what axis a given set of data from the table will use. These rules take sets of data and building oppositional relationships from the tables structure, using one-to-one relationships implicit in the structure to combine multiple set of opposed data into a single graph. The second set of rules are concerned with selecting the actual encoding type of a given set of data. These work very similarly to the APT system, selecting based on data type which encoding is best based on the rankings shown in Table 1.2.

However, context can limit which encodings are appropriate for a given instance of the information function. For example, if the display does not reproduce color, the color saturation and hue options are made unavailable. Additional statements about synthetic data, such as the knowledge that one set of data is the derivative of
another can lead to selecting a more compact encoding of the second set of data, such as slope arrows anchored to a line representing the first set.

However, the evaluation criterion of systems such as APT and SAGE are based on these concepts of representation design in the context of the written page. When we go outside the single page and into a space with varied display capability their rigid evaluation criterion no longer hold. These environmental contexts create situations in which the rules of formal graphic design fail. Graphic design texts such as Tufte’s universally value density of information in a graph. [Tufte, 1983][Tufte, 1990][Tufte, 1997] For example, user constraints such as proximity of display can effect how well the chart is drawn. If the user is far away from the available display a densely drawn graph encoding multiple data types with a complex key of the less accurately perceived encoding such as texture or shape will not be as useful as a set of simpler graph each encoding.

2.3 Communication Infrastructure

The communication infrastructure consists of three distinct components, information producers, consumers and managers. Information producers provide a facility for introducing new information containers into the system. Information consumers listen to the produced information and process it. This interaction can be seen in Figure 2.3. Information Managers serve in both capacities as well as providing a mechanism for storage and later requests for display of containers, shown in Figure 2.3.4.

2.3.1 Information Handling Agents

All agents that are capable of manipulating or communicating Information Containers must be info handling agents. These info handlers can handle errors in the system involving information containers and maintain a record of their owners and users. Information producers, consumers, and managers are all Information handling agents.
2.3.2 Information Producers

An information producer produces information containers, which it then communicates to interested parties. A producer allows consumers to register their interest in information of a specific type. If a consumer has registered itself as interested in any super type of the type of information container currently in production, then the producer will send the information container to the consumer.

2.3.3 Information Consumers

Information consumers are those entities capable processing information containers and registering with an information producer. A given consumer agent defines a behavior of the system with regard to an information container. For example, a Video consumer could register with an information producer controlling a VCR. The Video consumer could then manage the analog video output of the producer. Furthermore, a separate consumer could handle display of Text containers produced to report the status of the VCR.

2.3.4 Information Managers

The Information Manager, is the source of user interactions with the information display system, allowing user specification of display constraints and on demand display
Figure 2-3: Information Manager behavior: Here a manager is carrying out the role of a translation device, converting one type of information container to another type in order to meet a set of display constraints.

The information manager is also the placeholder for implementing complex display mechanisms. As the mechanism for requesting the display of information containers, it requires the capabilities of both a consumer and producer. A manager must receive information containers from producers in order to be aware that they are available for display. In addition, it must be able to call on a set of information consumers to utilize the display capacities of the system, thus working as an information producer. An example of typical manager behavior is included as figure 2.3.4

Managers are the prime location for context aware information translation, as a mechanism for display they allow the user to specify display constraints. However, in response to display requests they are able to decide between multiple consumers each of which might handle a suite of display devices. By allowing the specification of display constraints they are able to construct information functions on demand to adapt directly to the feedback from consumer’s and users’ desires.
Chapter 3

Implementation

This project's Information system manages information flow in the MIT AI lab's Intelligent Room. The high-level design was implemented accomplished through the use of Java and the Metaglue agent environment. The metaglue agent environment provides the foundation of the Intelligent Room.

3.1 Agents, Societies and Metaglue

The infosystem API is built on top of the agent architecture of the Metaglue development environment. Metaglue provides the underlying abstraction of agents and societies and a robust remote method invocation architecture which allows these agents to be distributed across many physical or virtual machines. [Phillips, 1999] [Michael Coen et al., 2000a]

In the application environment of the Intelligent room, societies represent a grouping of agents with a common abstract goal. The agents work on behalf of the society, and for the most part communicate within that society alone. The hal society for example contains all the agents associated with the task of running the intelligent office, named hal. Other societies exist on behalf of users, or different physical environments. [Peters, 2002]

Individual agents provide parts of the functionality described in Integrated In-
Figure 3-1: Infosystem Diagram: The implementation structure of the infosystem, showing the Java class interfaces and their implementing classes.
formation system example presented in section 1.6.2. In particular Metaglue agents provide the mechanics of natural language interpretation and web page display. Other agents detect and recognize gestures such as pointing. Many of these agents result in assertion of contexts or actions based on fusion of multiple modalities of input. For example, the pointing event created by Jim indicating the web-page is created by the assertion of the context of the language processing. The image processing might only guarantee that Jim is pointing at the web-page with 60% probability. However, given that the web-page was just mentioned can be used to eliminate other options, such as the wall or other displayed objects. [Michael Coen et al., 2000b]

3.2 Users, Information and Ownership

3.2.1 The Entity

An entity is simply described as a person, place or thing. Entities are to become the basis for the abstraction of ownership. An entity can represent an abstract service provider, a room, or user. In sum, an entity is anything that can be independently characterized as being able to possess objects or be provided services. This entity abstraction becomes crucial to providing the infosystem a mechanism for reasoning about ownership of information and tracking the agents working on behalf of a given entity. [Peters, 2002]

In the web-page to e-mail example, Jim, his boss, Bob the grant coordinator and the room itself are entities. Other entities such as individual agents work on behalf of these during the response. Throughout maintaining a knowledge of these relations allow additional information to be added to make the e-mailed URL more meaningful to the receiving party. That this message was sent on behalf of Jim’s boss to Bob from the room during this meeting provide Bob a host of information about how to approach the URL itself.
3.2.2 The Possessable Interface

The Possessable interface provides a specification through which the members of the infosystem are able to specify ownership of InfoContainers and Agents. Furthermore, a method `use(Entity e)`, is provided so that the behalf of relationship can be expressed. Using this method an entity can specify that it is using the possessable object and then later specify that it is done with the object using the `done()` method. Individual implementers of the possessable interface can specify different behaviors with regard to this method. For example an individual object could implement a security mechanism that only allowed method calls to the object to be made if it was in use by a specific entity.

Other possible uses of the concept of ownership of InfoContainers are the maintenance of confidentiality and copyright. An agent attempting to display a possible object could react to it differently adding the appropriate copyright symbols if it was owned by an entity other than the original owner. This mechanism allows the webpage that is passed on in the e-mail to include a proper reference of the originating group. This alerts Bob immediately to the context of the information, the email is owned by Jim and is sent on behalf of Jim’s boss but contains pertinent information to the grant process because the included web-page is from the group in direct competition for the grant.

3.2.3 Information Representation, Information Containers

3.2.4 InformationContainer

The InfoContainer class is the cornerstone of the information system. InfoContainers are possesable objects that encapsulate a given piece of information. They are strongly typed Java classes. The Java inheritance mechanism is used to build a hierarchy of information types. This type hierarchy allows for the development of general mechanisms for dealing with general categories of information.

This structure is directly relevant to the encapsulation and processing of the re-
Figure 3-2: The InfoContainer: The InfoContainer implements the possesable interface and provides a series of methods for comparison of Information types. The view method provides a mechanism for directly applying InfoFunctions to the InfoContainer.

```
Possesable
setPosition( Entity )
Entity getOwner()
Entity[] getUsers()
use( Entity )
done( Entity )
```

```
InfoContainer
InfoContainer( Entity owner )
String getDescription()
setIdDescription( String )
boolean isType( InfoContainer )
boolean isType( String type )
static String getType( Class )
String getExtendedType()
static String getExtendedType( String infoType )
InfoContainer view( InfoFunction )
throws InfoException
```

Figure 3-3: The InfoFunction: The InfoFunction provides the view method, taking an InfoContainer of some type and producing one of another.

```
InfoFunction
InfoContainer view( InfoContainer )
throws InfoException
```

quest for web-page to email conversion. Both the web-page and e-mail are types of information container, each representing an actual piece of concrete data. A WebPageContainer contains a URL and other relevant information about the web-page, while an EmailContainer contains the newly constructed e-mail. An information function defines how to encapsulate the information including in a WebPageContainer in a EmailContainer.
3.3 Translation Infrastructure, Information Functions

Because the Information container is a strongly typed Java object, they can be operated on by Java method calls. However, to retrain the integrity of the information contained, it is useful to structure these operators. Those individual information operators concerned with translation of information from one type to another are encapsulated by the abstraction of an InfoFunction object. These information functions provide a single public method, view(InfoContainer). This method takes in an InfoContainer as input and outputs another InfoContainer.

These InfoFunctions are defined as taking many input types and returning just one type of output. To implement this many to one relationship extenders of InfoFunction implement more specifically typed versions of the view method. When an information function is applied to an Information container, the most specific method for the given container type is automatically called. If no method is found the view method throws an InfoException to notify the caller of an error in translation.

In the demo example a WebPageContainer is converted into an EmailContainer using an ToEmail function. The ToEmail function defines a set of generic methods for extracting information from InfoContainers of several types and encapsulating this information in an EmailContainer. In the case of the WebPageContainer the function represents the web-page with a URL, and a set of context based ownership and usage relations with the entities which have handled the web-page. These are included in the body of an e-mail contained in the output EmailContainer.

3.4 Communication Infrastructure

Translation alone does not fulfill the goal of information management. It is necessary to provide a mechanism for communicating the information containers. This communication infrastructure ties the translation mechanism and the information container
3.4.1 Information Handling Agents

At the basis of the information communication system, is another possessable object, the InfoHandlingAgent. Each of these agents implements the possessable interface and is a metaglue agent. Furthermore, all information handling agents implement the `HandleError(InfoError)` method. This provides a universal mechanism for other members of the infosystem to report errors in communication or processing. An overview of the methods provided by the InfoHandling interfaces is seen in the following diagram 3.4.1.

3.4.2 Information Producers

Information container creation and broadcast is handled by the InfoProductionAgents. Individual producers of InfoContainers extend the InfoProducerAgent. The InfoProducerAgent provides two methods `addConsumer`, and `removeConsumer`. The `addConsumer` method allows information consumers to register their interest in a specified type of InfoContainer produced by the InfoProducer. The `removeCon-
Figure 3-5: The InfoProducer: the InfoProducer interface is implemented by the InfoProducerAgent. The InfoProducerAgent a set of useful protected methods to all subclasses.

<table>
<thead>
<tr>
<th>InfoProducer</th>
<th>InfoProducerAgent</th>
</tr>
</thead>
<tbody>
<tr>
<td>addConsumer( InfoConsumer, infoContainer )</td>
<td>protected produce( InfoContainer )</td>
</tr>
<tr>
<td>addConsumer( InfoConsumer, String InfoType )</td>
<td>protected addProducedType( String InfoType )</td>
</tr>
<tr>
<td>removeConsumer( InfoConsumer, infoContainer )</td>
<td>protected addProducedType( String info )</td>
</tr>
<tr>
<td>boolean produces( infoContainer )</td>
<td>protected addProducedType( Class )</td>
</tr>
<tr>
<td>boolean produces( String infoType )</td>
<td>protected addProducedType( Class )</td>
</tr>
<tr>
<td>Class[] getProducedTypes()</td>
<td>protected addProducedType( Class )</td>
</tr>
</tbody>
</table>

sumer method provides the dual functionality, allowing InfoConsumers to inform a InfoProducer that they are no longer interested in a type of information container. The InfoProducer automatically notifies the InfoConsumers registered for any type of Information Container. In addition, a producer notifies a consumer of all InfoContainers of the subtypes of the information type the consumer has registered for.

There is one discrepancy in the functionality of these two methods. When an InfoConsumer registers to listen for information production of an information type, it automatically receives the sub-types produced as well. It should be noted that when a removeConsumer( InfoConsumer, InfoType ) call is made that only the consumer is removed if and only if it was previously registered for exactly that information type.

InfoProducers provide the protected method produce( InfoContainer ) to subclasses allowing subclasses a simple interface to the broadcast mechanism. It automatically uses the NotifierAgent to transmit the produced Information container to those consumers listening for information of that type. This method was used by an agent monitoring the meeting to produce a WebPageContainer corresponding to the web-page Jim was pointing at.

### 3.4.3 Information Consumers

Information Consumers provide two main services, receiving InfoContainers directly using remote method invocation, and receiving InfoContainers broadcast by InfoProducers via Notifier. The consume(InfoContainer) method is implemented by sub-
Figure 3-6: The InfoConsumer: InfoConsumerAgent implements the InfoConsumer interface, providing protected methods for maintenance of the consumes methods.

<table>
<thead>
<tr>
<th>InfoConsumer</th>
<th>InfoConsumerAgent</th>
</tr>
</thead>
<tbody>
<tr>
<td>consume(InfoContainer)</td>
<td>protected addConsumedType(String infoType)</td>
</tr>
<tr>
<td>throws NotConsumedException</td>
<td>protected addConsumedType(InfoContainer info)</td>
</tr>
<tr>
<td>boolean consumes(String infoType)</td>
<td>protected addConsumedType(Class c)</td>
</tr>
<tr>
<td>Class[] getConsumedTypes()</td>
<td></td>
</tr>
</tbody>
</table>

classes to specify a behavior upon receiving an InfoContainer. If the InfoConsumer does not consume information of the type of InfoContainer, then the consume method will by default throw the NotConsumedException back to the caller. In the case of the consumption of broadcast InfoContainers, the InfoConsumer reports an error to the InfoProducer broadcasting by calling the handleError(InfoError) method on the producer.

This consumption mechanism provides the functionality for display or transmission of Jim's e-mail. A consumer of EmailContainers that was previously defined with Jim's e-mail account information is able to send the e-mail included in the EmailContainer on his behalf.

3.5 Information Managers

InfoManager agents serve several roles in the infosystem. First they implement both the InfoConsumer and InfoProducer Interfaces. Second, they provide the show(InfoContainer) method. In this capacity they are the placeholder for the logic of display. The show(InfoContainer, DisplayConstraint[]) method can be overridden to provide specific functionality for display of InfoContainers. In this role as a display arbitrator, the InfoManager accepts the Containers created by a producer and stores them for later display. The show method allows for the "user" to specify as series of abstract DisplayConstraints which can be tested on a given Display object and InfoContainer pair. Thus, to define a complex display behavior the programmer implements a set
Figure 3-7: The InfoManager: InfoManagerAgent implements the InfoManager interface, in order to maintain correctness of the broadcast receipt it extends InfoConsumerAgent. However, it implements the protected methods from InfoProducer for ease of extension.

---

### InfoProducer

- `addConsumer(InfoConsumer, InfoContainer)`
- `addConsumer(InfoConsumer, String InfoType)`
- `removeConsumer(InfoConsumer, InfoContainer)`
- `removeConsumer(InfoConsumer, String InfoType)`
- `boolean produces(InfoContainer)`
- `boolean produces(String InfoType)`
- `Class[] getProducedTypes()`

### InfoConsumer

- `consume(InfoContainer)` throws `NotConsumedException`
- `boolean consumes(InfoContainer)`
- `boolean consumes(String InfoType)`
- `Class[] getConsumedTypes()`

### InfoManager

- `show(InfoContainer)`
- `show(InfoContainer, DisplayConstraint[])`

### InfoManagerAgent

- `protected produce(InfoContainer)`
- `protected produce(InfoConsumer, InfoContainer)`
- `protected addProducedType(String infoType)`
- `protected addProducedType(InfoContainer info)`
- `protected addProducedType(Class c)`
- `protected addProducedType(Class)`
of DisplayConstraints, a logic bundle for the show method and the consume method.

In the demo meeting, several InfoManagers are at work. A secretary like Information Manager works on behalf of Bob to maintain information about where Bob is. Another such agent works on behalf of the room providing information about the schedule of meetings in the room, allowing the determination that Bob is the grant coordinator Jim is referring to. Furthermore Jim has his own manager which interprets his request for the display of the web-page with the constraint that it be displayed to the grant coordinator. This manager does most of the work. Upon receiving the request to display the WebPageContainer, it first queries the room's manager for Bob's identity, then Bob's manager for Bob's location. Next using its own expert system it then decides to display the WebPageContainer to Bob as an EmailContainer, setting up and executing the translation. Finally it finds Jim's EmailConsumer and feeds it the translated information. The EmailConsumer then sends the Email to Bob.

### 3.5.1 The MetaManagerAgent

In the course of executing this example Jim's manager was able to make specific queries about what consumers and managers where available. In order to make this possible each society runs a MetaManager. The MetaManager is an implementation of the most generic of all InfoManagers. The MetaManager provides a utility for monitoring the InfoHandlingAgents in any given society. Upon creation all InfoHandlingAgents are registered with the MetaManager. The MetaManager provides methods for Agents to query the active InfoHandlingAgents. In addition the MetaManager provides a limited Graphic User Interface (GUI) that allows user monitoring and re-direction of the flow of InfoContainers. Seen in Figure 3.5.1 this GUI would allow Jim to send the WebPageContainer to an his manager manually even in the absence of the more complex multi-modal interaction tools.
Figure 3-8: MetaManager GUI. This GUI provides direct interaction to the infosystem.

This GUI provides direct interaction to the infosystem. The InfoContainer and active InfoHandlers are all listed. In addition, the Feed button allows manual user direction of any InfoContainer to any InfoConsumer. The Link button allows the user to connect InfoConsumers to InfoProducers automatically. Registration of all the appropriate containers and producers for the consumer is automatically facilitated.
Chapter 4

Conclusion

Corporate America faces a dilemma as we move toward a technical labor economy based on short term rather than long term employee commitment. Old-fashioned tools for creating corporate memory are breaking down in a world with a more mobile workforce. Modern automated systems can help to preserve design rationale and other parts of corporate memory often limited to the oral tradition.

However, these automated information gathering systems are often more effort than they are worth. While attempts to decrease the amount effort required to input information into these systems have met with limited but promising success, little has been done to make the process of extracting and understanding information from these systems easier.

The ability to provide adaptive systems of representation will allow the user to more easily understand information stored by automated information gathering systems. The creation of a software tool that facilitates the understanding of information makes large scale information systems a more valuable tool for preserving corporate memory.

The information system presented here gives future researchers working with the Intelligent room a important tool. It allows the integration of the promising contextual input tools with the automated representation tools. Using this system produces more complex information display behaviors than have been previously possible.
Appendix A

Code

This appendix presents the Java Code for the basic abstractions of the infosystem. These abstractions and their use is described in detail in the implementation section of this thesis.
A.1 Possessable

```java
package infosystem;

import metaglue.*;
import java.rmi.*;

public interface Possessable {
    // An abstract interface describing an agent which
    // can interact with the infosystem

    /********
    * Returns the Entity this belongs to
    *******/

    public Entity getOwner() throws RemoteException;

    /********
    * Tries to set the owner of this to Entity e
    *******

    public void setOwner(Entity e) throws RemoteException;

    /********
    * Entity e wants to use is use this object.
    *******

    public void use(Entity e) throws RemoteException;

    /********
    * Returns the Entities currently using this
    *******

    public Entity[] getUsers() throws RemoteException;

    /********
    * Entity e no longer wants to use this.
    *******

    public void done(Entity e) throws RemoteException;
}
```
package infosystem;
import java.util.*;
import java.rmi.*;
import infosystem.functions.*;
import java.io.*;

public class InfoContainer implements Possessable, Serializable{
    Entity owner;
    Vector users;

    //this is an entirely arbitrary description used to distinguish
    //different InfoContainers of the same type.
    String description;
    // the formal name of this infotype
    String infoType;
    // the extended formal name of this infotype
    String extendedType;

    public InfoContainer( Entity owner ]{
        this.owner = owner;
        this.users = new Vector();
        this.description = this.getClass().toString();
        infoType = this.getClass().toString();
        infoType = infoType.substring(infoType.indexOf(' ') + 1, infoType.length());
        extendedType = createClassTree();
    }

    public Entity getOwner() throws RemoteException{
        return owner;
    }

    public void setOwner(Entity e) throws RemoteException{
        // do nothing... statically owned entity
    }

    public void use( Entity e ) throws RemoteException{
        // no protection access protection at all
        if(! users.contains(e) )
            users.add(e);
    }

    public Entity[] getUsers() throws RemoteException{
        Entity[] ret = new Entity[users.size()];
        users.copyInto(ret);
        return ret;
    }
}
public void done( Entity e ) throws RemoteException{
    users.remove(e);
}

global String getDescription(){
    return description;
}

global void setDescription( String description ){
    this.description = description;
}

global boolean isType(InfoContainer info){
    Class c = info.getClass();
    return this.getClass().isAssignableFrom(c);
}

global boolean isType(String type){
    try{
        Class c = Class.forName(type);
        return this.getClass().isAssignableFrom(c);
    }catch(ClassNotFoundException e){
        return false;
    }
}

global String getType(){
    return infoType;
}

static global String getType( Class c, String infoType )
throws ClassNotFoundException
{
    return createClassTree(Class.forName(infoType));
}

public String getExtendedType(){
    //returns a notifier compatable extended type
    return extendedType;
}

private String createClassTree(){
    Class c = this.getClass();
    return createClassTree( c );
}

static public String getExtendedType( String infoType )
    throws ClassNotFoundException
{
    return createClassTree(Class.forName(infoType));
}
static final char separator = ' , '; // used to separate individual classes
static final char dot_char = ' * '; // used to separate individual parts of a class

static private String createClassTree(Class c) {
    String ret = "";
    Vector superclasses = new Vector();
    try{
        Class baseCase = Class.forName("info\system\InfoContainer");
        while( baseCase.isAssignableFrom(c)){
            String infoType = c.toString();
            infoType = infoType.substring(infoType.indexOf(' ') + 1, infoType.length());
            infoType = infoType.replace('.', dot_char);
            superclasses.add(infoType);
            c = c.getSuperclass();
        }
        Enumeration e = superclasses.elements();
        while( e.hasMoreElements() ){
            ret += (String) e.nextElement() + separator;
        }
        ret = ret.substring(0,ret.length() - 1);
        return ret;
    }catch(ClassNotFoundException e){
        ret = c.toString();
        ret = ret.substring(ret.indexOf(' ') + 1, ret.length());
        return ret;
    }
}

public InfoContainer view(InfoFunction lamda) throws InfoException {
    // Returns an InformationContainer containing a view of the
    // information contained in this, as specified by lamda.
    return lamda.view(this);
}
package infosystem;

import java.lang.reflect.*;
import infosystem.functions.*;

public class InfoFunction {

    /**
     * composes this function g(x) with the input function f(x)
     * returns f(g(x))
     **/

    public InfoFunction compose( InfoFunction fun ) {
        return new ComposedFunction( this, fun );
    }

    /**
     * view function operates on the information in input and puts
     * returns an output of specific type, this throws the
     * InformationException if there is no method for creating a view
     * from an InformationContainer of the type of input to the type
     * InformationContainer of output ...
     *
     * ***** NB: DON'T reimplement apply( InfoContainer i ) *****
     * Subclasses should implement view( InputType i ) where InputType is a
     * subclass of InfoContainer:
     *
     **/

    public InfoContainer view( InfoContainer input ) throws InfoException {
        // this next bit tries to do the right thing even if the most
        // general invocation of this is called...
        // find the most specific method of this and invoke it...

        try{
            Class function = this.getClass();
            System.err.println( "InfoFunction says: function = " + function.getName() );

            Method[] methods = function.getMethods();
            Class current = input.getClass();

            System.err.println( "InfoFunction says: input = " + current.getName() );

            //look for a method that is most specific...
            Method method = null;
            boolean loop = true;

            //...
while(loop){
    String name = current.getName();
    System.err.println("InfoFunction says: looping on = "+current.getName());

    for(int i = 0; loop && i < methods.length; i++){
        Method m = methods[i];
        if( m.getName().equals("view") ){
            Class[] params = m.getParameterTypes();
            if( params.length == 1 && name.equals(params[0].getName()) ){
                method = m; loop = false;
            }
        }
    }
    current = current.getSuperclass();
    loop = loop && !name.equals("infosystem.InfoContainer");
}
if( method == null )
    throw new InfoException( function.getName() +
        " cannot be used to compute a view of "+
        input.getClass().getName());
if( method.getDeclaringClass().getName().equals("infosystem.InfoFunction") )
    //prevent dispatching back to this method...
    throw new InfoException( function.getName() +
        " cannot be used to compute a view of "+
        input.getClass().getName());

Object[] in = {input};
System.err.println("InfoFunction invoking method: "+method);
return (InfoContainer) method.invoke(this, in);
} catch( Exception e ){
    throw( new InfoException(e));
    }
}
A.4 InfoHandling

package infosystem;

import metaglue.*;
import java.rmi.*;
import infosystem.containers.*;
import agentland.util.*;

public interface InfoHandling extends Possessable, Good{
    //handles errors
    public void handleError(InfoError error)
        throws RemoteException;
}

A.5 InfoHandlingAgent

package infosystem;

import metaglue. *
import java.rmi. *
import java.util. *
import infosystem.containers. *
import infosystem.managers. *
import agentland.util. *

public abstract class InfoHandlingAgent extends GoodAgent implements InfoHandling {
    // An agent which can interact with the infosystem
    Entity owner;
    Vector users;

    public InfoHandlingAgent() throws RemoteException {
        setLogLevel(LogStream.DEBUG);
        log("DEBUG", "Starting an InfoHandlingAgent");
        if(! (this instanceof MetaManager )){
            log("DEBUG", "Registering this with MetaManagerAgent");
            MetaManager mm =
                (MetaManager) reliesOn("infosystem.managers.MetaManager");
            mm.register(this);
            owner = new Entity((new Attribute("owner")).getValue());
            users = new Vector();
        }

        public void handleError( InfoError error ) throws RemoteException { }

        public String getChannel( AgentID broadcaster, String infoType, String channelType ) throws ClassNotFoundException, 
        throws ClassNotFoundException {
            String channel = broadcaster.toString();
            channel += "." + channelType + ".";
            channel += InfoContainer.getExtendedType(infoType);
            return channel;
        }

    }

}
InfoContainer info,
String channelType )
{
    /*
     * The standard channel types are PRODUCTION
     * PRODUCTION channels carry outgoing InfoContainers from
     * the InfoProducers
     */

    String channel = broadcaster.toString();
    channel += "." + channelType + ".";
    channel += info.getExtendedType();
    return channel;
}

protected void finalize() throws Throwable{
    // here is the clean up with the metamanager tied into the java virtual machine
    try{
        if (! (this instanceof MetaManager )){
            MetaManager mm =
            (MetaManager) reliesOn("info system.managers.MetaManager");
            mm.unRegister(this);
        }
    } catch(Exception e){}
    super.finalize();
}

public Entity getOwner() throws RemoteException;
    return owner;
}

public void setOwner(Entity e) throws RemoteException;
    // To implement security subclasses should over ride this method.
    owner = e;
}

public void use( Entity e ) throws RemoteException;
    // no protection access protection at all ;)
    if(! users.contains(e ))
        users.add(e);
}

public Entity[] getUsers() throws RemoteException{
    Entity[] ret = new Entity[users.size()];
    users.copyInto(ret);
    return ret;
}

public void done( Entity e ) throws RemoteException{
    users.remove(e);
}

public boolean equals( Object o ) {
    log("DEBUG", "Entering the infoHandler equals method with : "+ o);
}
if(! (o instanceof InfoHandling ))
    return false;
try{
    return ((InfoHandling) o).getAgentID().equals(getAgentID());
}catch( Exception e ){ return false; }
public interface InfoConsumer extends InfoHandling {
    // An abstract interface describing an agent which can "consume" the
    // information in a container...

    public void consume(InfoContainer info)
        throws RemoteException, NotConsumedException;

    public boolean consumes(InfoContainer ic)
        throws RemoteException;

    public boolean consumes(String infoType)
        throws RemoteException;

    public Class[] getConsumedTypes()
        throws RemoteException;
}
A.7 InfoConsumerAgent

```java
package infosystem;

import metaglue.*;
import java.rmi.*;
import java.util.*;
import infosystem.containers.*;
import agentland.util.*;
import java.io.*;

public class InfoConsumerAgent extends InfoHandlingAgent implements InfoConsumer {
    // An abstract interface describing an agent which can "consume" the
    // information in a container...

    Vector consumables;

    public InfoConsumerAgent() throws RemoteException{
        consumables = new VectorQ;
    }

    /*
     * "Feeds" this information consumer the information in the container ic
     */
    public void consume ( InfoContainer info )
        throws RemoteException, NotConsumedException
    {
        //default behavior is to report an error to the caller
        throw new NotConsumedException(info, this);
    }

    /*
     * Allows the InfoConsumer to be fed through notifier
     */
    public void tell( Secret s )
        throws RemoteException{
        //default behavior: decodes the message and consumes the infocontainer within
        // tell the producer to handle an error if we can't consume the
        // container
        Serializable details = s.details();
        if(!(details instanceof InfoContainer)){
            log("ERROR", "Someone tried to feed ";
            getAgentID() + "; a non-InfoContainer ");
        }
        InfoContainer info = (InfoContainer) details;
        try{
            consume(info);
        }catch( NotConsumedException e ){
            InfoHandling producer = (InfoHandling) reliesOn(s.source);
            producer.handleError( new InfoError( getAgentID(),
        }
    }
}
```
protected void addConsumedType( String infoType ){
    try{
        Class c = Class.forName(infoType);
        addConsumedType(c);
    }catch(ClassNotFoundException e){
        log("ERROR", "Attempted to add a non-existent infotype for consumption");
    }
}

protected void addConsumedType( InfoContainer info ){
    Class c = info.getClass();
    addConsumedType(c);
}

protected void addConsumedType( Class c ){
    log("DEBUG","Adding Consumed Type of Class "+c);
    consumables.add(c);
}

public Class[] getConsumedTypes() throws RemoteException{
    Class[] ret = new Class[consumables.size()];
    consumables.toArray(ret);
    return ret;
}

private boolean consumes(Class c){
    Enumeration e = consumables.elements();
    while( e.hasMoreElements() ){
        Class el = (Class) e.nextElement();
        if(el.isAssignableFrom(c))
            return true;
    }
    return false;
}

public boolean consumes( InfoContainer info ) throws RemoteException{
    return consumes(info.getClass());
}

public boolean consumes( String infoType ) throws RemoteException{
    try{

log("DEBUG", "Getting class for String infoType.");
Class c = Class.forName(infoType);
log("DEBUG", "String infoType is Class " + c);
boolean ret = consumes(c);
log("DEBUG", "String infoType consumable? " + ret);
return ret;
}catch(ClassNotFoundException e){
    log("ERROR", "String infoType is Class not found!");
    return false;
}
package infosystem;

import metaglue.*;
import java.rmi.*;
import infosystem.consumers.*;
import infosystem.containers.*;

public interface InfoProducer extends InfoHandling {
    // An abstract interface describing an agent which
    // can "produce" a container of information ...

    public void addConsumer(InfoConsumer consumer, InfoContainer info)
               throws RemoteException;
    public void addConsumer(InfoConsumer consumer, String infoType)
               throws RemoteException;
    public void removeConsumer(InfoConsumer consumer, InfoContainer info)
               throws RemoteException;
    public void removeConsumer(InfoConsumer consumer, String infoType)
               throws RemoteException;
    public boolean produces(InfoContainer info)
               throws RemoteException;
    public boolean produces(String infoType)
               throws RemoteException;
    public Class[] getProducedTypes()
               throws RemoteException;
}
A.9 InfoProducerAgent

```java
package infosystem;

import metaglue.*;
import agentland.util.*;
import java.rmi.*;
import java.util.*;
import infosystem.consumers.*;
import infosystem.containers.*;
import infosystem.managers.*;

public class InfoProducerAgent extends InfoHandlingAgent implements InfoProducer{
    private Vector produce; // Vector of the string defs for the infocontainer types this produces
    private MetaManager mm;

    public InfoProducerAgent() throws RemoteException{
        log("DEBUG", "Starting InfoProducer Agent");
        produce = new VectorQ;
        mm = (MetaManager) reliesOn( "infosystem. managers. MetaManager" );
        addConsumer( mm , "infosystem. InfoContainer" );
    }

    private void addConsumer( AgentID consumer, String infoType ) throws RemoteException {
        log("DEBUG", "Adding " + consumer + " to as a consumer of " + infoType );
        try{
            String productionChannel = getChannel( getAgentID(), infoType, "PRODUCTION");
            notifier.addSpy( consumer, productionChannel );
            log("DEBUG", consumer + " added");
        }catch( ClassNotFoundException e ){
            log("ERROR", consumer + " not added. InfoContainers of type " + infoType + "don't exist");
        }
    }

    private void removeConsumer( AgentID consumer, String infoType ) throws RemoteException {
        log("DEBUG", "Removing " + consumer + " as a consumer of " + infoType );
        try{
            String productionChannel = getChannel( getAgentID(), infoType, "PRODUCTION");
        }
    }
}
```
notifier.removeSpy(consumer, productionChannel);
log("DEBUG", consumer + " removed");
}
catch(ClassNotFoundException e){
log("ERROR", consumer + " not removed. InfoContainers of type " + infoType + " don't exist");
}

public void addConsumer(InfoConsumer consumer, InfoContainer info) throws RemoteException {
addConsumer(consumer.getAgentID(), info.getType());
consumer.use(new Entity(getAgentID()));
this.use(new Entity(consumer.getAgentID()));
}

public void addConsumer(InfoConsumer consumer, String infoType) throws RemoteException {
addConsumer(consumer.getAgentID(), infoType);
consumer.use(new Entity(getAgentID()));
this.use(new Entity(consumer.getAgentID()));
}

public void removeConsumer(InfoConsumer consumer, InfoContainer info) throws RemoteException {
removeConsumer(consumer.getAgentID(), info.getType());
consumer.done(new Entity(getAgentID()));
this.done(new Entity(consumer.getAgentID()));
}

public void removeConsumer(InfoConsumer consumer, String infoType) throws RemoteException {
removeConsumer(consumer.getAgentID(), infoType);
consumer.done(new Entity(getAgentID()));
this.done(new Entity(consumer.getAgentID()));
}

public boolean produces(InfoContainer info) throws RemoteException {
return produces(info.getClass());
}

public boolean produces(String infoType) throws RemoteException {
// by default doesn't produce anything
try{
return produces(Class.forName(infoType));
}
catch(ClassNotFoundException e){
return false;
}

private boolean produces( Class c ) throws RemoteException{
    // by default doesn't produce anything
    Enumeration e = produce.elements();
    while( e.hasMoreElements() ){
        Class producedClass = (Class) e.nextElement();
        if(producedClass.isAssignableFrom(c))
            return true;
    }
    return false;
}

protected void produce( InfoContainer info ) throws RemoteException{
    // a convenience method for subclasses to use for broadcast
    // Do not use this except from in subclasses, it must be
    // public to use from threads in subclasses but really should
    // be protected
    // if(! produces(info) ) { several behaviors possible }
    String channel = getChannel( getAgentID(), info, "PRODUCTION");
    Secret s = new Secret( getAgentID(), channel, info );
    notifier.notify(s);
}

protected void produce( InfoConsumer consumer, InfoContainer info )
    throws RemoteException, NotConsumedException
{
    // produce an info container for only the ears of one consumer. Using this ensures
    // that infocontainers produced by this manager are seen by the MetaManager
    consumer.consume(info);
    mm.consume(info);
}

protected void addProducedType( String infoType ){
    try{
        Class c = Class.forName(infoType);
        addProducedType(c);
    }catch(ClassNotFoundException e){
        log("ERROR", "Attempted to add a non-existant infotype for consumption");
    }
}

protected void addProducedType( InfoContainer info ){
    Class c = info.getClass();
    addProducedType(c);
}

protected void addProducedType( Class c ){
    log("DEBUG", "Adding produced info type: " + c);
    produce.add(c);
    log("DEBUG", "Added produced info type: " + produce.lastElement());
}
public Class[] getProducedTypes() throws RemoteException{
    Class[] ret = new Class[produce.size()];
    produce.toArray(ret);
    return ret;
}
package infosystem;

import java.rmi.*;

public interface InfoManager extends InfoHandling, InfoProducer, InfoConsumer {

    /**
     * Attempts to show the Person this is acting on behalf of the information
     * in the InfoContainer info
     *
     * assumes that this can show the info container to the user this
     * is acting on behalf of, and that errors in display are completely
     * handled.
     *
     * params info The container holding the information to show
     */
    public void show(InfoContainer info) throws RemoteException;

    /**
     * Attempts to show the information in the InfoContainer info
     * given a set of constraints on the display...
     *
     * params info The container holding the information to show
     */
    public void show(InfoContainer info, DisplayConstraint[] constraints) throws RemoteException;
}
A.11 InfoManagerAgent

package infosystem;

import java.rmi.*;
import java.util.*;
import metaglue.*;
import infosystem.managers.*;
import agentland.util.*;

abstract public class InfoManagerAgent extends InfoConsumerAgent implements InfoManager {
    private Vector produce;
    MetaManager mm;

    /*
     * Provides the basic structure for a user to interact with
     * information display
     */

    public InfoManagerAgent() throws RemoteException{
        produce = new VectorQ;
        if( !( ( this instanceof MetaManager) || ( this instanceof MetaManagerAgent))) {
            mm = (MetaManager) reliesOn("info system.managers.MetaManager");
            addConsumer( mm,"info system.InfoContainer");
        }
    }

    public void show( InfoContainer info ) throws RemoteException{
        // attempts to show the owner the information in the InfoContainer info
        // Cowardly does nothing until Entity.where() exists...
    }

    public void show( InfoContainer info, DisplayConstraint[] constraints ) 
        throws RemoteException
    {
        // Attempts to show the information in the InfoContainer info
        // given the constraints.
    }

    *********
    * Implementation of methods from infosystem.producers.InfoProducer
    *********

    private void addConsumer( AgentID consumer, String infoType ) 
        throws RemoteException
    {
        log("DEBUG", "Adding " + consumer + " to as a consumer of " + infoType );
        try{
            String productionChannel = getChannel( getAgentID(), infoType, "PRODUCTION");
            notifier.addSpy( consumer, productionChannel );
        }
    }

}
log("DEBUG", consumer + " added");
} catch (ClassNotFoundException e) {
    log("ERROR", consumer + " not added. InfoContainers of type " + infoType + " don't exist");
}

private void removeConsumer(AgentID consumer, String infoType) throws RemoteException {
    log("DEBUG", "Removing " + consumer + " as a consumer of " + infoType);
    try{
        String productionChannel = getChannel(getAgentID(), infoType, "PRODUCTION");
        notifier.removeSpy(consumer, productionChannel);
        log("DEBUG", consumer + " removed");
    } catch (ClassNotFoundException e) {
        log("ERROR", consumer + " not removed. InfoContainers of type " + infoType + " don't exist");
    }
}

public void removeConsumer(InfoConsumer consumer, String infoType) throws RemoteException {
    removeConsumer(consumer.getAgentID(), infoType);
    consumer.done(new Entity(getAgentID()));
    this.done(new Entity(consumer.getAgentID()));
}

public void addConsumer(InfoConsumer consumer, String infoType) throws RemoteException {
    addConsumer(consumer.getAgentID(), infoType);
    consumer.use(new Entity(getAgentID()));
    this.use(new Entity(consumer.getAgentID()));
}

public void addConsumer(InfoConsumer consumer, InfoContainer info) throws RemoteException {
    addConsumer(consumer.getAgentID(), info.getType());
    consumer.use(new Entity(getAgentID()));
    this.use(new Entity(consumer.getAgentID()));
}

public void removeConsumer(InfoConsumer consumer, InfoContainer info) throws RemoteException {
    removeConsumer(consumer.getAgentID(), info.getType());
    consumer.use(new Entity(getAgentID()));
    this.use(new Entity(consumer.getAgentID()));
}
public boolean produces(InfoContainer info) throws RemoteException{
    return produces(info.getClass());
}

public boolean produces(String infoType) throws RemoteException{
    try{
        return produces(Class.forName(infoType));
    }catch(ClassNotFoundException e){
        return false;
    }
}

private boolean produces(Class c) throws RemoteException{
    Enumeration e = produce.elements();
    while( e.hasMoreElements() ){
        Class el = (Class) e.nextElement();
        if(el.isAssignableFrom(c))
            return true;
    }
    return false;
}

protected void produce(InfoContainer info) throws RemoteException{
    // a convenience method for subclasses to use for broadcast to consumers which
    // have expressed an interest in this
    String channel = getChannel(getAgentID(), info, "PRODUCTION");
    Secret s = new Secret(getAgentID(), channel, info);
    notifier.notify(s);
}

protected void produce(InfoConsumer consumer, InfoContainer info) throws RemoteException, NotConsumedException{
    // produce an info container for only the ears of one consumer. Using this ensures
    // that infocontainers produced by this manager are seen by the MetaManager
    consumer.consume(info);
    mm.consume(info);
}

protected void addProducedType(String infoType){
    try{
        addProducedType(Class.forName(infoType));
    }catch(ClassNotFoundException e){
        log("ERROR", "Attempted to add a non-existent infotype for consumption");
    }
}

protected void addProducedType(InfoContainer info){

addProducedType(info.getClass());
}

protected void addProducedType(Class c)
{
produce.add(c);
}

public Class[] getProducedTypes() throws RemoteException{
Class[] ret = new Class[produce.size()];
consumables.toArray(ret);
return ret;
}
A.12 MetaManager

```
package infosystem.managers;

import infosystem.*;
import infosystem.containers.*;
import infosystem.consumers.*;
import infosystem.producers.*;
import java.rmi.*;

public interface MetaManager extends InfoManager {
    // register is automatically called on creation of an InfoHandlingAgent
    public void register(InfoHandling handler) throws RemoteException;
    // unregister is called upon InfoHandlingAgent destruction
    public void unregister(InfoHandling handler) throws RemoteException;
    public InfoConsumer[] getConsumers() throws RemoteException;
    public InfoProducer[] getProducers() throws RemoteException;
    public InfoManager[] getManagers() throws RemoteException;
    public InfoConsumer[] getConsumers(InfoContainer info) throws RemoteException;
    public InfoProducer[] getProducers(InfoContainer info) throws RemoteException;
    public InfoManager[] getManagers(InfoContainer info) throws RemoteException;
    public InfoConsumer[] getConsumers(String infoType) throws RemoteException;
    public InfoProducer[] getProducers(String infoType) throws RemoteException;
    public InfoManager[] getManagers(String infoType) throws RemoteException;
}
```
A.13 MetaManagerAgent

package infosystem.managers;

import java.rmi.*;
import java.util.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.table.*;
import javax.swing.event.*;
import metaglue.*;
import infosystem.*;
import infosystem.containers.*;
import infosystem.producers.*;
import infosystem.consumers.*;

public class MetaManagerAgent extends InfoManagerAgent implements MetaManager {
    Vector consumers;
    Vector producers;
    Vector managers;
    Frame producerFrame;
    Frame consumerFrame;
    Frame containerFrame;
    Frame managerFrame;
    Frame functionFrame;
    Frame toolboxFrame;
    ContainerModel containerModel;
    FunctionModel functionModel;
    HandlingModel producerModel;
    HandlingModel consumerModel;
    HandlingModel managerModel;
    InfoContainer selectedContainer;
    InfoConsumer selectedConsumer;
    InfoProducer selectedProducer;

    public MetaManagerAgent() throws RemoteException{
        log("INFO", "Starting up the MetaManager");

        consumers = new Vector();
        producers = new Vector();
        managers = new Vector();

        // listen in on all the producer broadcasts
addConsumedType("infosystem.InfoContainer");

//populate these three vectors from attributes, that are comma seperated strings
//of agentnames
// attribute consumers
// attribute producers
// attribute managers
StringTokenizer parser = new StringTokenizer(getAttribute( "consumers" ), "," );
log("INFO", "Starting InfoConsumers");
while( parser.hasMoreTokens() ){
    String consumer = parser.nextToken();
    log("INFO", "Registering InfoConsumer " + consumer );
    try{
        InfoConsumer ic = (InfoConsumer) reliesOn( consumer );
        register(ic);
    }catch(Exception e){
        log("ERROR", "MetaManagerAgent: failed starting " + consumer);
    }
}
parser = new StringTokenizer(getAttribute( "producers" ), "," );
log("INFO", "Starting InfoProducers");
while( parser.hasMoreTokens() ){
    String producer = parser.nextToken();
    try{
        InfoProducer ip = (InfoProducer) reliesOn( producer );
        register(ip);
    }catch(Exception e){
        log("ERROR", "MetaManagerAgent: failed starting " + producer);
    }
}
parser = new StringTokenizer(getAttribute( "managers" ), "," );
log("INFO", "Starting InfoManagers");
while( parser.hasMoreTokens() ){
    String manager = parser.nextToken();
    try{
        InfoManager im = (InfoManager) reliesOn( manager );
        register(im);
    }catch(Exception e){
        log("ERROR", "MetaManagerAgent: failed starting " + manager);
    }
}

// Setup GUI
log("DEBUG", "Starting MetaManager GUI");

producerFrame = new Frame();
consumerFrame = new Frame();
containerFrame = new Frame();
managerFrame = new Frame();
functionFrame = new Frame();
toolboxFrame = new Frame();
containerModel = new ContainerModel();
functionModel = new FunctionModel();
producerModel = new HandlingModel();
consumerModel = new HandlingModel();
managerModel = new HandlingModel();

// initial selections in the GUI
selectedContainer = null;
selectedConsumer = null;
selectedProducer = null;

log("DEBUG", "Starting Status Windows");
setupStatusWindows();
log("DEBUG", "Starting Toolbox Window");
setupToolBox();

log("DEBUG", "Making it all visible");
toolboxFrame.setVisible(true);
producerFrame.setVisible(false);
consumerFrame.setVisible(false);
containerFrame.setVisible(false);
managerFrame.setVisible(false);
functionFrame.setVisible(false);

}

public void register( InfoHandling handler ) throws RemoteException{
    if( handler instanceof InfoManager ){
        if( !managers.contains( handler ) ){
            managers.add( handler );
            managerModel.updateModel( handler, true );
        }
    }else
    if( handler instanceof InfoProducer ){
        if( !producers.contains( handler ) ){
            producers.add( handler );
            producerModel.updateModel( handler, true );
        }
    }else
    if( handler instanceof InfoConsumer ){
        if( !consumers.contains( handler ) ){
            consumers.add( handler );
            consumerModel.updateModel( handler, true );
        }
    }else
    log("ERROR", "Attempted to register Unknown InfoHandlerType");
}

public void unregister( InfoHandling handler ) throws RemoteException{
    if( handler instanceof InfoManager ){
        managerModel.updateModel( handler, false );
    }else
    71
if (handler instanceof InfoProducer) {
    producerModel.updateModel(handler, false);
} else {
    if (handler instanceof InfoConsumer) {
        consumerModel.updateModel(handler, false);
    } else {
        log("ERROR", "Attempted to Un-register Unknown InfoHandlerType");
    }
}

public InfoConsumer[] getConsumers() throws RemoteException {
    InfoConsumer[] cons = new InfoConsumer[consumers.size()];
    consumers.copyInto(cons);
    return cons;
}

public InfoProducer[] getProducers() throws RemoteException {
    InfoProducer[] pros = new InfoProducer[producers.size()];
    producers.copyInto(pros);
    return pros;
}

public InfoManager[] getManagers() throws RemoteException {
    InfoManager[] mans = new InfoManager[managers.size()];
    managers.copyInto(mans);
    return mans;
}

public InfoConsumer[] getConsumers(InfoContainer info) throws RemoteException {
    Enumeration cons = consumers.elements();
    Vector tmp = new Vector();
    while (cons.hasMoreElements()) {
        InfoConsumer consumer = (InfoConsumer) cons.nextElement();
        if (consumer.consumes(info)) {
            tmp.add(consumer);
        }
    }
    InfoConsumer[] ret = new InfoConsumer[tmp.size()];
    tmp.copyInto(ret);
    return ret;
}

public InfoProducer[] getProducers(InfoContainer info) throws RemoteException {
    Enumeration pros = producers.elements();
    Vector tmp = new Vector();
    while (pros.hasMoreElements()) {
        InfoProducer producer = (InfoProducer) pros.nextElement();
        if (producer.produces(info)) {
            tmp.add(producer);
        }
    }
    InfoProducer[] ret = new InfoProducer[tmp.size()];
    tmp.copyInto(ret);
    return ret;
}
public InfoProducer[] getProducers( String infoType) throws RemoteException{
    Vector tmp = new Vector();
    Enumeration pros = producers.elements();
    while(pros.hasMoreElements()){
        InfoProducer producer = (InfoProducer) pros.nextElement();
        if( producer.produces(infoType)) {
            tmp.add(producer);
        }
    }
    InfoProducer[] ret = new InfoProducer[tmp.size()];
    tmp.copyInto(ret);
    return ret;
}

public InfoConsumer[] getConsumers( String infoType) throws RemoteException{
    Vector tmp = new Vector();
    Enumeration cons = consumers.elements();
    while(cons.hasMoreElements()) {
        InfoConsumer consumer = (InfoConsumer) cons.nextElement();
        if( consumer.consumes(infoType)) {
            tmp.add(consumer);
        }
    }
    InfoConsumer[] ret = new InfoConsumer[tmp.size()];
    tmp.copyInto(ret);
    return ret;
}
tmp.add(consumer);
}

InfoConsumer[] ret = new InfoConsumer[tmp.size()];
tmp.copyInto(ret);
return ret;
}

public InfoManager[] getManagers( String infoType ) throws RemoteException{
    Enumeration mans = managers.elements();
    Vector tmp = new Vector();
    while(mans.hasMoreElements()){
        InfoManager manager = (InfoManager) mans.nextElement();
        if( manager.produces(infoType) || manager.consumes(infoType) ){
            tmp.add(manager);
        }
    }
    InfoManager[] ret = new InfoManager[tmp.size()];
tmp.copyInto(ret);
return ret;
}

public void consume( InfoContainer info ) throws RemoteException, NotConsumedException{
    ContainerRecord record = new ContainerRecord(info);
    containerModel.addRecord(record);
}

abstract class TableModel extends AbstractTableModel {
    final protected int START_NUM_ROWS = 3;
    protected String[] columnNames;
    protected int nextEmptyRow;
    protected int numRows;
    protected Vector data;

    public TableModel( ) {
        columnNames = new String[0];
        nextEmptyRow = 0;
        numRows = 0;
        data = new Vector();
    }

    public String getColumnName(int column) {
        if(column >= columnNames.length )
            return "";
        return columnNames[column];
    }
public synchronized int getColumnCount() {
    return columnNames.length; //NUM_COLUMNS
}

public int getRowCount(){
    return numRows;
}

public void addRecord( Object record ){
    // this is unsafe if called with anything but the expected
    // record type
    numRows++;
    data.add(record);
    fireTableRowsInserted(nextEmptyRow, nextEmptyRow);
    nextEmptyRow++;
}

public synchronized void clear() {
    int oldNumRows = numRows;
    numRows = START_NUM_ROWS;
    data.removeAllElements();
    nextEmptyRow = 0;

    if (oldNumRows > START_NUM_ROWS) {
        fireTableRowsDeleted(START_NUM_ROWS, oldNumRows - 1);
    }
    fireTableRowsUpdated(0, START_NUM_ROWS - 1);
}

class ContainerRecord {
    InfoContainer info;
    Date time;

    ContainerRecord( InfoContainer container ){
        info = container;
        time = new Date(System.currentTimeMillis());
    }

    String getType(){
        return info.getType();
    }
    Date getProductionTime(){
        return time;
    }
    String getOwner(){
        try{
            return info.getOwner().toString();
        }
        catch(Exception e){
            return "";
        }
    }
    InfoContainer getContainer(){
        return info;
    }
}

class FunctionRecord {
    InfoFunction fun;
}
String type;

FunctionRecord( InfoFunction fun ){
    this.fun = fun;
    type = fun.getClass().toString();
}

String getType(){ return type; }
}

class ContainerModel extends TableModel {
    //uses ContainerRecords
    ContainerModel(){
        String[] cols = {"Type", "Production Time", "Owner"};
        columnNames = cols;
    }

    public InfoContainer getContainer( int row ){
        ContainerRecord cr = (ContainerRecord) data.elementAt(row);
        return cr.getContainer();
    }

    public synchronized Object getValueAt(int row, int column) {
        try {
            ContainerRecord cr = (ContainerRecord) data.elementAt(row);
            switch (column) {
                case 0:
                    return cr.getType();
                case 1:
                    return cr.getProductionTime();
                case 2:
                    return cr.getOwner();
            }
        } catch (Exception e) {
            log("DEBUG", "ERROR Recalling that col: " + column + " row: " + row);
        }
        return "-";
    }
}

class HandlingRecord {
    InfoHandling agent;
    boolean active;

    HandlingRecord( InfoHandling agent, boolean active ){ 
        this.agent = agent;
        this.active = active;
    }

    String getAgentID(){
        try{
            AgentID id = agent.getAgentID();
        } catch (Exception e) {
            log("DEBUG", "ERROR Recalling that col: " + column + " row: " + row);
        }
        return "-";
    }
}
public synchronized Object getValueAt(int row, int column) {
    try {
        HandlingRecord handler = (HandlingRecord) data.elementAt(row);
        switch (column) {
        case 0:
            return handler.getAgentID();
        case 1:
            return handler.getStatus();
        }
    } catch (Exception e) {} 
    return "";
}

public InfoHandling getHandler(int row) {
    HandlingRecord record = (HandlingRecord) data.get(row);
    return (InfoHandling) record.getStub();
}

public void updateModel(InfoHandling handler, boolean active) {
    try{
        HandlingRecord record = new HandlingRecord(handler, active);
        if( ! data.contains(record) ){
            addRecord(record);
        } else{
            int index = data.indexOf(record);
            HandlingRecord old_record = (HandlingRecord) data.get(index);
            data.removeElementAt(index);
            data.insertElementAt(record, index);
            fireTableRowsUpdated(index, index);
        }
    } catch(Exception e){
    }
}
```java
class FunctionModel extends TableModel {
    FunctionModel()
    {
        String[] cols = { "Type" };  
        columnNames = cols;
    }

    public synchronized Object getValueAt(int row, int column) {
        try {
            FunctionRecord fun = (FunctionRecord) data.elementAt(row);
            switch (column) {
                case 0:
                    return fun.getTypeQ;
            }
        } catch (Exception e) {}
        return "";
    }
}

JTable setupContainerTable(){
    JTable table = new JTable(containerModel);
    table.setColumnSelectionAllowed(false);
    table.setCellSelectionEnabled(false);
    ListSelectionModel rowSM = table.getSelectionModel();
    rowSM.addListSelectionListener(new ListSelectionListener() {
        public void valueChanged(ListSelectionEvent e) {
            //Ignore extra messages.
            if (e.getValueIsAdjusting()) return;

            ListSelectionModel lsm = (ListSelectionModel)e.getSource();
            if (lsm.isSelectionEmpty()) {
                log("DEBUG", "No rows are selected.");
                selectedContainer = null;
            } else {
                int selectedRow = lsm.getMinSelectionIndex();
                selectedContainer = containerModel.getContainer( selectedRow );
                log("DEBUG", "Selected a container: " + selectedContainer);
            }
        }
    });
    return table;
}

JTable setupProducerTable(){
    JTable table = new JTable(producerModel);
    table.setCellSelectionEnabled(false);
    table.setColumnSelectionAllowed(false);
    ListSelectionModel rowSM = table.getSelectionModel();
    ListSelectionModel rowSM = table.getSelectionModel();
    ListSelectionModel rowSM = table.getSelectionModel();
```

78
rowSM.addListSelectionListener(new ListSelectionListener()
    {
    public void valueChanged(ListSelectionEvent e)
    {
        // Ignore extra messages.
        if (e.getValueIsAdjusting()) return;

        ListSelectionModel Ism = (ListSelectionModel)e.getSource();
        if (ism.isSelectionEmpty())
        {
            log("DEBUG", "No rows are selected.");
            selectedProducer = null;
        } else {
            int selectedRow = ism.getMinSelectionIndex();
            selectedProducer =
                (InfoProducer) producerModel.getHandler( selectedRow );
            log("DEBUG", "Selected a producer: " + selectedProducer);
        }
    }
});
return table;

JTable setupConsumerTable()
{
    JTable table = new JTable(consumerModel);
    table.setCellSelectionEnabled(false);
    table.setColumnSelectionAllowed(false);
    ListSelectionModel rowSM = table.getSelectionModel();
    rowSM.addListSelectionListener(new ListSelectionListener()
        {
        public void valueChanged(ListSelectionEvent e)
        {
            // Ignore extra messages.
            if (e.getValueIsAdjusting()) return;

            ListSelectionModel Ism = (ListSelectionModel)e.getSource();
            if (ism.isSelectionEmpty())
            {
                log("DEBUG", "No rows are selected.");
                selectedConsumer = null;
            } else {
                int selectedRow = ism.getMinSelectionIndex();
                selectedConsumer =
                    (InfoConsumer) consumerModel.getHandler( selectedRow );
                log("DEBUG", "Selected a consumer: " + selectedConsumer);
            }
        }
    });
    return table;
}

JTable setupManagerTable()
{
    // There is no selection behavior for this as that would result in
    // ambiguous behavior

    JTable table = new JTable(managerModel);
    table.setCellSelectionEnabled(false);
table.setColumnSelectionAllowed(false);
return table;
}

JTable setupFunctionTable(){
  // There is no selection behavior for this as that would result in
  // ambiguous behavior
  JTable table = new JTable(functionModel);
table.setCellSelectionEnabled(false);
table.setColumnSelectionAllowed(false);
return table;
}

void setupStatusWindows() throws RemoteException{
  log("DEBUG","Setting Up Producer Table");
  JTable producerTable = setupProducerTable();
  log("DEBUG","Setting Up Consumer Table");
  JTable consumerTable = setupConsumerTable();
  log("DEBUG","Setting Up Container Table");
  JTable containerTable = setupContainerTable();
  log("DEBUG","Setting Up ManagerTable");
  JTable managerTable = setupManagerTable();
  log("DEBUG","Setting Up Function Table");
  JTable functionTable = setupFunctionTable();

  log("DEBUG","Setting Up Scroll Panes");
  JScrollPane producerPane = new JScrollPane(producerTable);
  JScrollPane consumerPane = new JScrollPane(consumerTable);
  JScrollPane containerPane = new JScrollPane(containerTable);
  JScrollPane managerPane = new JScrollPane(managerTable);
  JScrollPane functionPane = new JScrollPane(functionTable);

  log("DEBUG","Adding Panes to Frames");
  producerFrame.add(producerPane);
  consumerFrame.add(consumerPane);
  containerFrame.add(containerPane);
  managerFrame.add(managerPane);
  functionFrame.add(functionPane);

  producerFrame.setSize(producerFrame.getPreferredSize());
  producerFrame.setTitle(getAgentID().toString() + ": InfoProducers");
  consumerFrame.setSize(consumerFrame.getPreferredSize());
  consumerFrame.setTitle(getAgentID().toString() + ": InfoConsumers");
  containerFrame.setSize(containerFrame.getPreferredSize());
  containerFrame.setTitle(getAgentID().toString() + ": Produced InfoContainers");
  managerFrame.setSize(managerFrame.getPreferredSize());
  managerFrame.setTitle(getAgentID().toString() + ": InfoManagers");
  functionFrame.setSize(functionFrame.getPreferredSize());
  functionFrame.setTitle(getAgentID().toString() + ": InfoFunctions");
}
JRadioButton containerButton;
JRadioButton consumerButton;
JRadioButton producerButton;
JRadioButton managerButton;
JRadioButton functionButton;
JButton feedButton;
JButton linkButton;

void setupToolBox(){
    feedButton = new JButton("Feed"); // button to feed infocontainers to consumers
    linkButton = new JButton("Link"); // button to link consumers and producers

    containerButton = new JRadioButton("Show Containers");
    consumerButton = new JRadioButton("Show Consumers");
    producerButton = new JRadioButton("Show Producers");
    managerButton = new JRadioButton("Show Managers");
    functionButton = new JRadioButton("Show Functions");

    ItemListener listener = new ItemListener() {
        public void itemStateChanged(ItemEvent e) {
            Object source = e.getItemSelectable();
            if (source == containerButton) {
                containerFrame.setVisible(!containerFrame.isVisible());
            } else if (source == producerButton) {
                producerFrame.setVisible(!producerFrame.isVisible());
            } else if (source == consumerButton) {
                consumerFrame.setVisible(!consumerFrame.isVisible());
            } else if (source == managerButton) {
                managerFrame.setVisible(!managerFrame.isVisible());
            } else if (source == functionButton) {
                functionFrame.setVisible(!functionFrame.isVisible());
            }
        }
    }

    containerButton.addItemListener(listener);
    consumerButton.addItemListener(listener);
    producerButton.addItemListener(listener);
    managerButton.addItemListener(listener);
    functionButton.addItemListener(listener);

    BorderLayout layout = new BorderLayout();
    toolboxFrame.setLayout(layout);

    ActionListener actionListener = new ActionListener() {
        private Class[] intersect(Class[] consumed, Class[] produced) {
            Vector intersection = new Vector();
            for (int i = 0; i < produced.length; i++)
                for (int j = 0; j < consumed.length; j++)
                    if (consumed[j].isAssignableFrom(produced[i]))
                        intersection.add(produced[i]);
            return (Class[]) intersection.toArray();
        }
        
        public void actionPerformed(ActionEvent e) {
            // handle action
        }
    }

    buttonMap.put("Feed", feedButton);
    buttonMap.put("Link", linkButton);
    buttonMap.put("Containers", containerButton);
    buttonMap.put("Consumers", consumerButton);
    buttonMap.put("Producers", producerButton);
    buttonMap.put("Managers", managerButton);
    buttonMap.put("Functions", functionButton);
    buttonMap.put("Sort", sortButton);
    buttonMap.put("Filter", filterButton);
    buttonMap.put("Revert", revertButton);
    buttonMap.put("Clear", clearButton);
    buttonMap.put("Open", openButton);
    buttonMap.put("Save", saveButton);
    buttonMap.put("Add", addButton);
    buttonMap.put("Remove", removeButton);
    buttonMap.put("Move", moveButton);
    buttonMap.put("Search", searchButton);
    buttonMap.put("Help", helpButton);
}

BorderLayout layout = new BorderLayout();
toolboxFrame.setLayout(layout);

ActionListener actionListener = new ActionListener() {
    private Class[] intersect(Class[] consumed, Class[] produced) {
        Vector intersection = new Vector();
        for (int i = 0; i < produced.length; i++)
            for (int j = 0; j < consumed.length; j++)
                if (consumed[j].isAssignableFrom(produced[i]))
                    intersection.add(produced[i]);
        return (Class[]) intersection.toArray();
    }

    public void actionPerformed(ActionEvent e) {
        // handle action
    }
}

buttonMap.put("Feed", feedButton);
buttonMap.put("Link", linkButton);
buttonMap.put("Containers", containerButton);
buttonMap.put("Consumers", consumerButton);
buttonMap.put("Producers", producerButton);
buttonMap.put("Managers", managerButton);
buttonMap.put("Functions", functionButton);
buttonMap.put("Sort", sortButton);
buttonMap.put("Filter", filterButton);
buttonMap.put("Revert", revertButton);
buttonMap.put("Clear", clearButton);
buttonMap.put("Open", openButton);
buttonMap.put("Save", saveButton);
buttonMap.put("Add", addButton);
buttonMap.put("Remove", removeButton);
buttonMap.put("Move", moveButton);
buttonMap.put("Search", searchButton);
buttonMap.put("Help", helpButton);

81
if(!intersection.contains(produced[i])){
    log("DEBUG", "Found intersection: " + produced[i].toString());
    intersection.add(produced[i]);
}

Class[] ret = new Class[intersection.size()];
intersection.toArray(ret);
return ret;

public void actionPerformed(ActionEvent e) {
    if (e.getActionCommand().equals("feed")) {
        InfoConsumer consumer = selectedConsumer;
        InfoContainer container = selectedContainer;
        log("DEBUG", "Trying to feed " + container + " to " + consumer);
        if( !(consumer == null && container == null )){
            log("DEBUG", "Trying to feed " + container + " to " + consumer);
            try{
                consumer.consume(container);
                log("DEBUG", "Fed " + container + " to " + consumer);
            } catch( Exception exception ){
                log("ERROR", "Can't feed: " + consumer + " with " + container);
            }
        } else if (e.getActionCommand().equals("link")){
            InfoConsumer consumer = selectedConsumer;
            InfoProducer producer = selectedProducer;
            if( !(consumer == null && producer == null ) ){
                try{
                    Class[] consumedTypes = consumer.getConsumedTypes();
                    Class[] producedTypes = consumer.getProducedTypes();
                    Class[] intersection = intersect(consumedTypes, producedTypes);
                    for( int i = 0; i < intersection.length; i++ ){
                        String infoType = InfoContainer.getType(intersection[i]);
                        log("DEBUG", "Found type of intersection: ***:+" + infoType + ":***");
                        producer.addConsumer(consumer, infoType);
                    }
                } catch(Exception exception){
                    log("ERROR", "Failed to link selected consumer and producer");
                }
            }
        }
    }
}

feedButton.setActionCommand("feed");
linkButton.setActionCommand("link");
feedButton.addActionListener(actionListener);
linkButton.addActionListener(actionListener);
Panel actions = new Panel();
actions.add(feedButton);
actions.add(linkButton);

actions.setLayout(new GridLayout(1,2));
actions.doLayout();

Panel buttons = new Panel();
buttons.add(containerButton);
buttons.add(consumerButton);
buttons.add(producerButton);
buttons.add(managerButton);
buttons.add(functionButton);

buttons.setLayout(new GridLayout(5,1));
buttons.doLayout();

toolboxFrame.add(buttons, BorderLayout.CENTER);
toolboxFrame.add(actions, BorderLayout.SOUTH);
toolboxFrame.doLayout();

toolboxFrame.setSize( toolboxFrame.getPreferredSize() );
} }

83
Bibliography


